Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires

U.B.A. FACULTAD DE INGENIERÍA

Departamento de Informática
75.10 – TÉCNICAS DE DISEÑO
TRABAJO PRÁCTICO

Curso: 2012 – 2er Cuatrimestre

GRUPO N° 14	
APELLIDO, Nombres	**N° PADRÓN**
Servetto, Matías	91363
Schenkelman, Damián	90728
Rodriguez, Sebastian	90202
Fecha de Aprobación :	
Calificación :	
Firma de Aprobación :	

Observaciones:

Contenido

Vistas 4+1

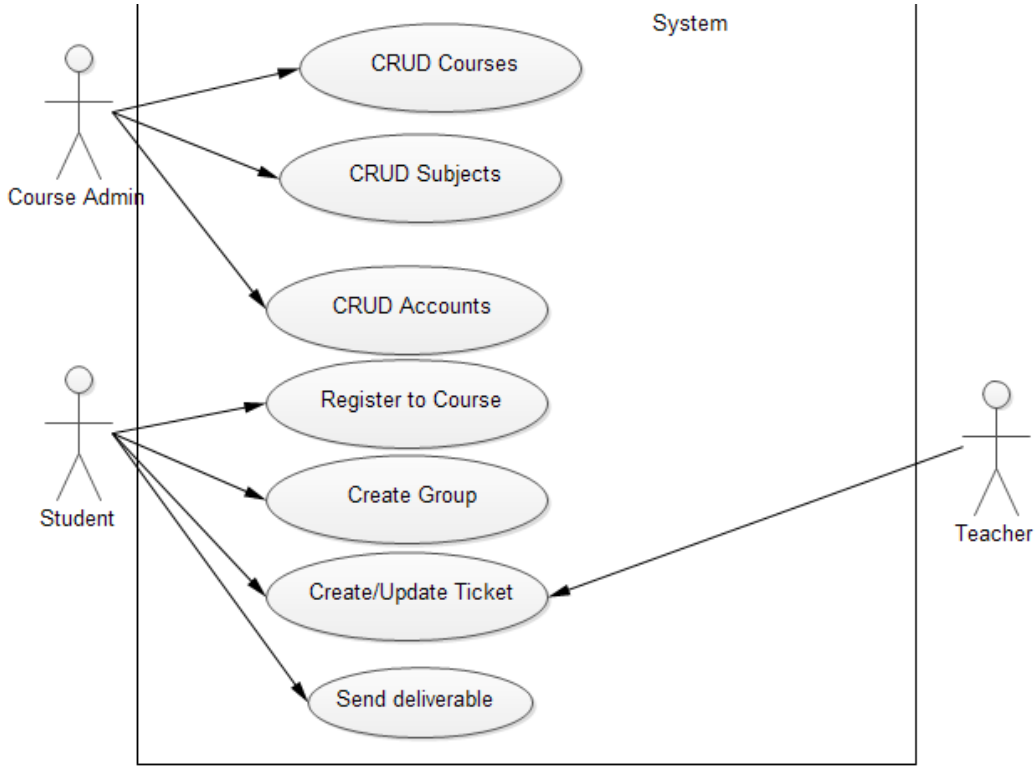
Vista de Casos de Uso

En esta sección se mencionan los casos de uso principales de la aplicación.

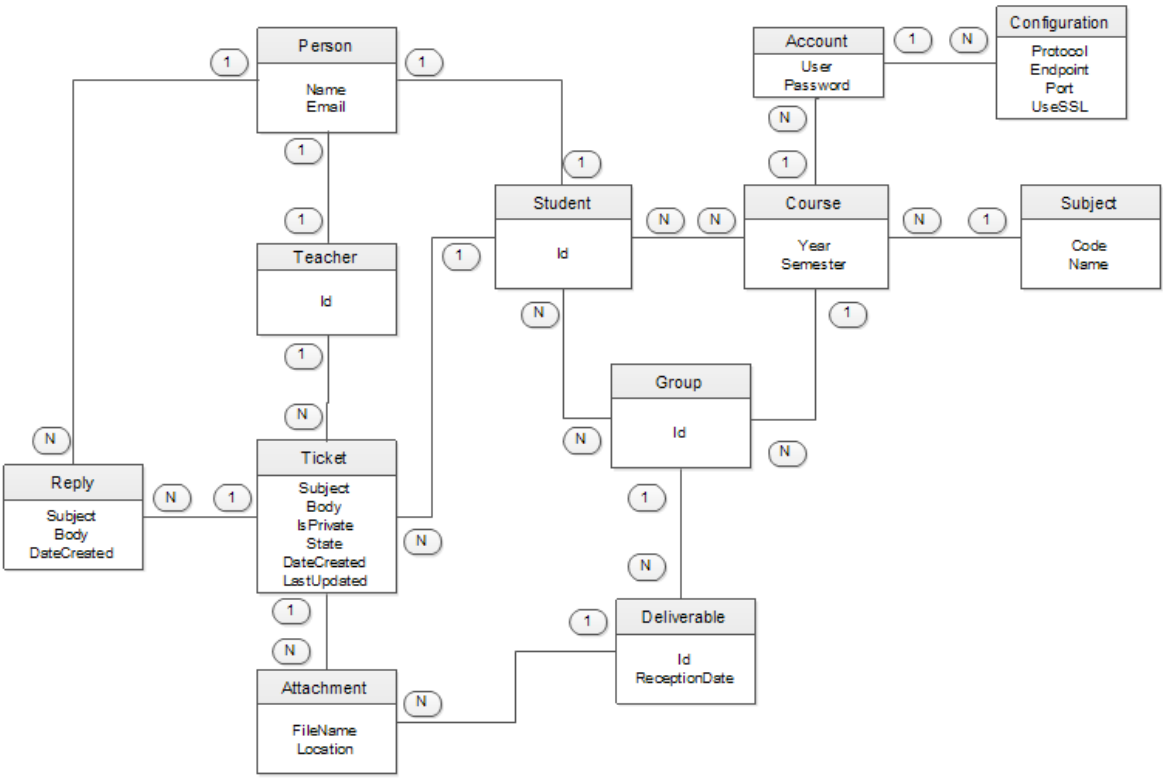
1. Configurar cuentas de e-mail de un curso de una materia.
2. Administrar materias y cursos (crear, modificar y eliminar).
3. Procesamiento de e-mails bajo determinadas reglas, las cuales permiten:
    - Dar de alta de alumnos en un curso.
    - Registrar entregables de alumnos, incluyendo la persistencia en disco de archivos adjuntos a e-mails.
    - Crear nuevos tickets.
    - Responder a tickets existentes.
    - Crear grupos en un curso y agregar alumnos al grupo.
4. Monitoreo de estado de los temas de discusión.
5. Ver estadísticas (alumnos inscriptos, grupos asignados, TPs recibidos, e-mails recibidos, estados de temas de discusión, número de e-mails que no cumplen con las reglas).

Nota: Los ítems 4 y 5 no son parte del scope del proyecto (al menos en esta versión). Es decir, la información necesaria para cumplir con dichos casos de uso será guardada, pero el sistema no permitirá consultarla.

El siguiente diagrama de casos de uso contempla aquellos que están dentro del alcance del proyecto.



Adicionalmente, incluimos a continuación una versión simplificada del diagrama de entidad-interrelación, ya que consideramos que puede ser útil para facilitar la comprensión de los casos de uso solicitados por la aplicación.



Vista de Desarrollo

Para el desarrollo de la aplicación se utilizara:

- C# como lenguaje de programación.
- Visual Studio 2010 como ambiente de desarrollo.
- Entity Frameworkcomo ORM.
- Moq y Moles para crear mocks.
- Unity como contenedor de inyección de dependencias.
- Una biblioteca a definir para la obtención de e-mail a través de POP3.
- ASP.NET MVC 4 para el sitio web que permite administrar cursos, materias, cuentas y tickets.

Vista Lógica

La siguiente figura muestra una vista en capas de la aplicación.

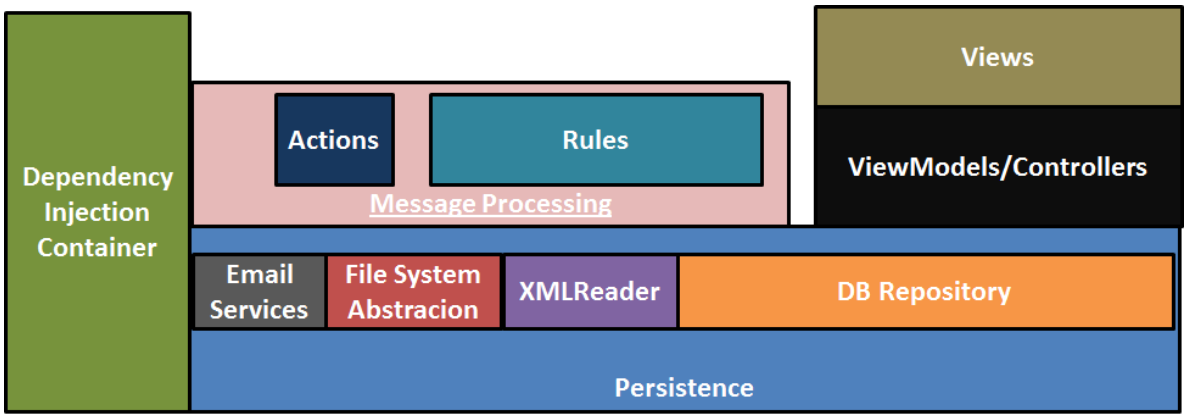
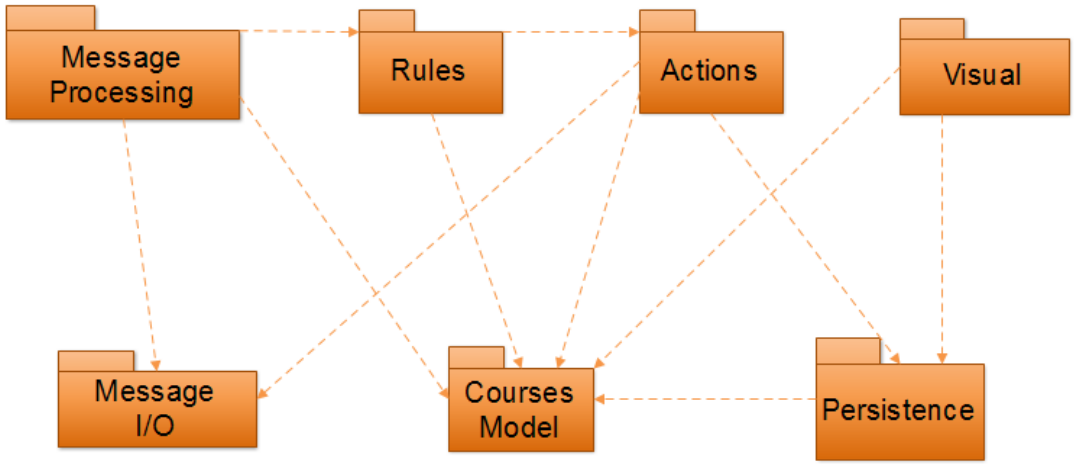


Diagrama de paquetes

Los siguientes son los diagramas de paquetes resultantes pensados para la aplicación (sin incluir los paquetes de pruebas).

Este es el diagrama que muestra los paquetes más relevantes y sus relaciones de utilización:



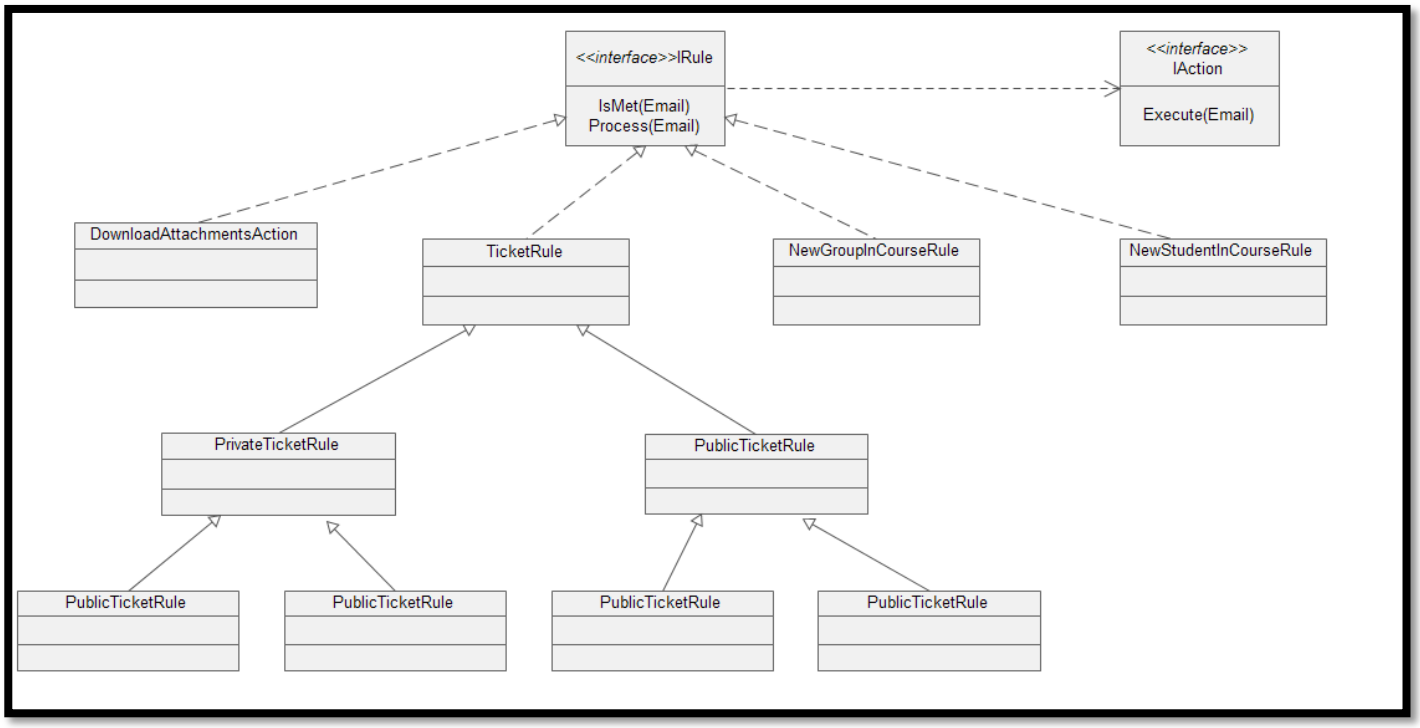
El contenido de cada paquete se explica a continuación:

- **CoursesModel:**Contiene a las clases que representan al dominio de la aplicación.
- **Persistence:**Contiene las clases necesarias para guardar los datos generados por el procesamiento de e-mails y recuperarlos.
- **Visual:**Contiene las interfaces de las vistas y la implementación de las mismas así como también implementación de los controladores para cada vista y las interfaces de los mismos.
- **MessageProcessing:** Contiene a los componentes necesarios para procesar los mensajes recuperados a través de las diferentes reglas.
- **Message I/O:** Contiene a los componentes necesarios para enviar y recibir mensajes.

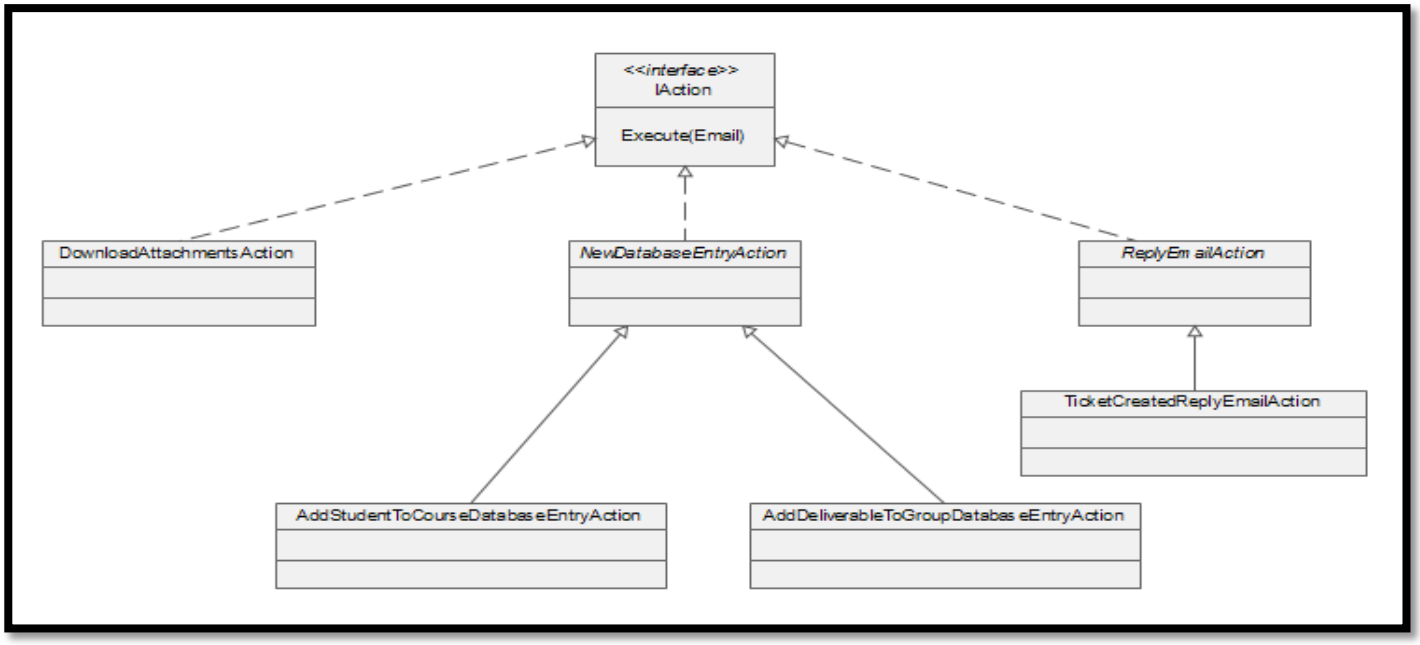
Diagramas de Clases

A continuación se muestran los diagramas de clases diseñados para la aplicación.

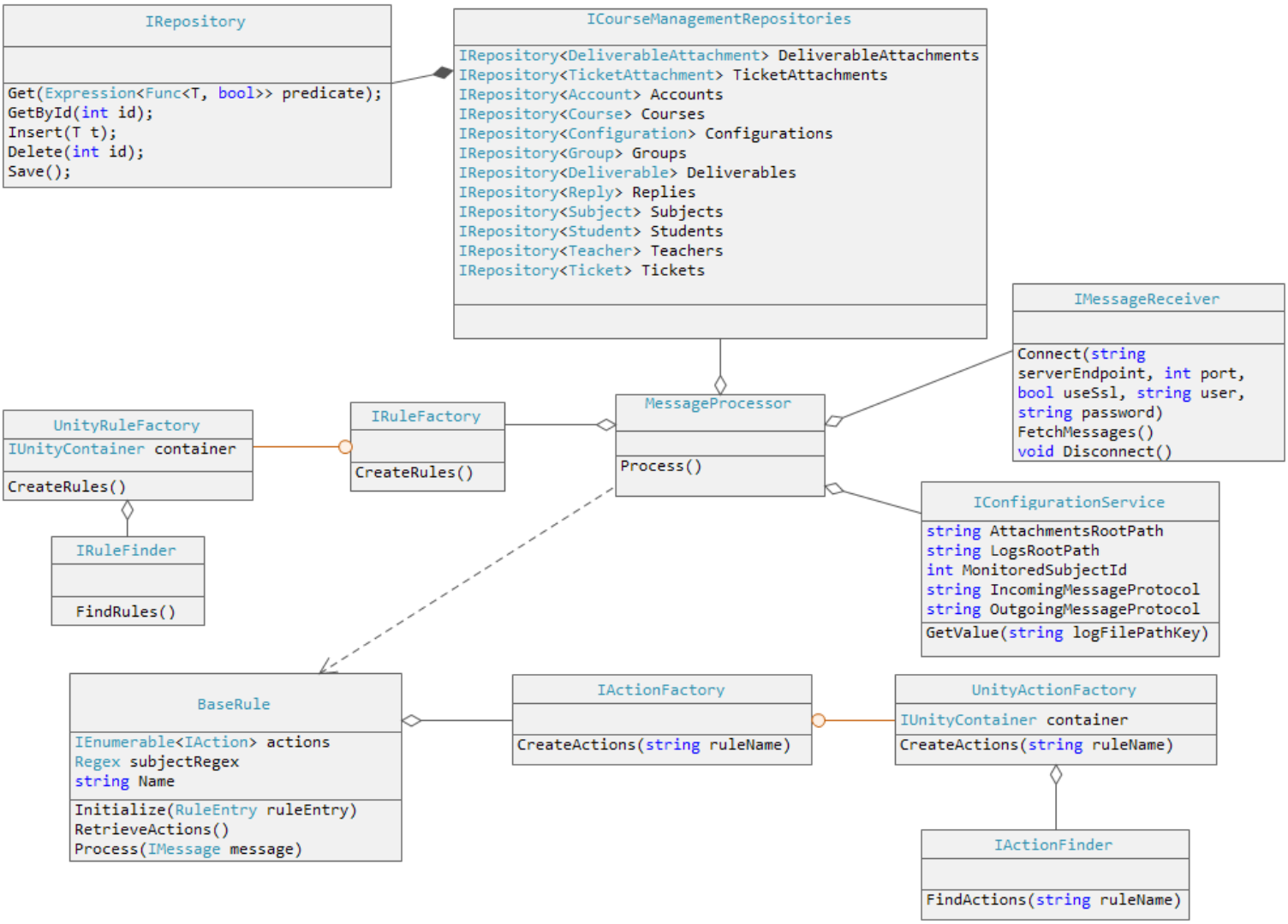
Se muestran dos diagramas parciales, que contienen las jerarquías de Reglas y Acciones; y por último se muestra el diagrama de clases general.



(Diagrama de Clases Parcial – Jerarquía de Reglas)



(Diagrama de Clases Parcial – Jerarquía de Acciones)



(Diagrama de Clases General)

Diagramas de Secuencia

Se detallan los diagramas más relevantes de la aplicación. Los mismos corresponden a las secuencias de obtención de reglas y acciones a partir del procesado de emails.

Nota: Si bien la lógica del procesado de mensajes es más compleja, se decidió no reflejarla sobre los diagramas por una cuestión de simplicidad debido a que no aporta valor y presta a confusión sobre el flujo principal.

(RuleBootstrapSequence)

(ActionBootstrapSequence)

Vista de procesos

La aplicación cuenta con:

- Un proceso en un loop infinito (worker)que continuamente recuperara los e-mails nuevos de las cuentas configuradas para una materia y los procesará.
- Un proceso para una aplicación de web, que permitirá adminstrar materias, cursos, cuentas de e-mail y configuración de cuentas.

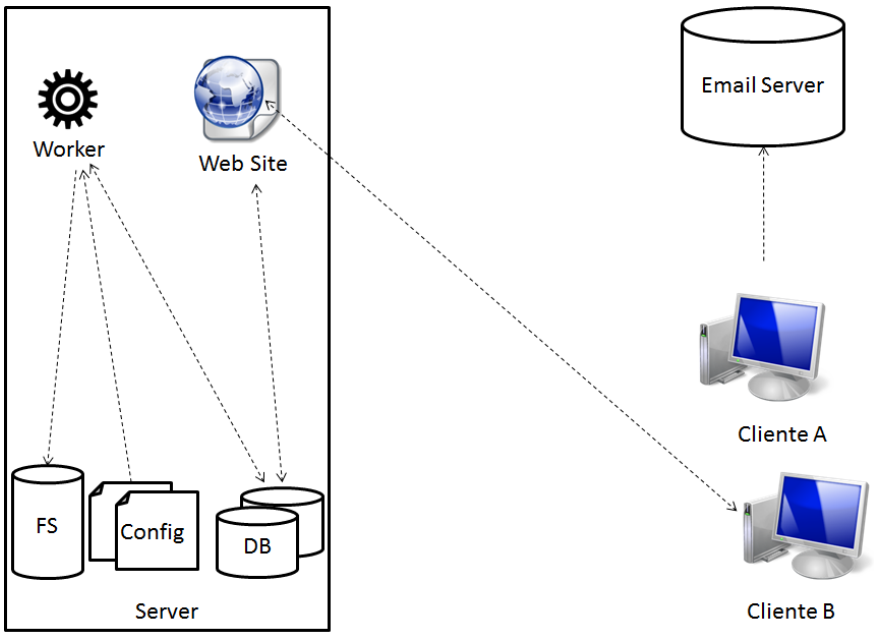
Estos dos procesos no necesitan mecanismos de sincronización ya que en ningún momento interactúan directamente entre sí, sino que escriben la base de datos.

Nota: El código de la aplicación web, que permite administrar las distintas configuraciones y datos ingresados, está en su mayoría autogenerado. La idea de la inclusión de dicho apartado en la última etapa fue puramente estética para dar una idea de la visión a futuro que se tenía en cuanto al alcance de la aplicación. Por lo tanto, al ser un overdeliver, no posee tests y no sigue buenas prácticas.

Vista física

A continuación se muestra la distribución propuesta para la aplicación suponiendo una interfaz web para la administración de cursos. Otra posibilidad (no ilustrada aquí por ser más simple), involucra a una aplicación de escritorio que suplante a la interfaz web.

Nota: Esta distribución considera la demo de la aplicación (por cuestiones de tiempo y más importante aún, presupuestarias). Si esta aplicación fuera puesta en producción se recomendaría utilizar servidores separados para el Worker, la aplicación Web y la base de datos, así como también utilizar un servicio de Storage as a Service.



Decisiones de Diseño

Desacoplamiento mediante abstracciones

Desde el comienzo del proceso de desarrollo se busco trabajar con abstracciones, no implementaciones concretas, a fin de aumentar el desacople entre componentes, lo que provee mayor facilidad para cambiar los mismos y realizar tests unitarios.

Un ejemplo del desacoplamiento buscado se ve en la persistencia. La persistencia a los diferentes medios se realiza mediante abstracciones. En la aplicación desarrollada, los repositorios ejecutan consultas contra una base de datos. Implementando correctamente la interfaz de los mismos, se podría conseguir actualizar la aplicación para utilizar Google Spreadsheets.

De igual forma, se utiliza un patrón de "presentación separada", no para utilizar el mismo controller/viewmodel para distintas vistas, sino para mantener la lógica desacoplada de las vistas.

Para lograr el desacoplamiento de componentes, cada clase recibe en su constructor interfaces, no implementaciones concretas. Para facilitar la construcción de estos objetos y poder modificar de manera simple y rápida la implementación a utilizar, se decidió utilizar un DependencyInjectionContainer, UnityContainer. El mismo se configura mediante un Bootstrapper.

Procesado de Mensajes

Para el procesado de los mensajes, se decidió tomar como referencia el patrón "*Chain of Responsability*"; el patrón en concreto no fue implementado, pero la idea base del patrón fue la clave para lograr el correcto procesado de mensajes.

La decisión por la cual no se implementó dicho patrón fue por un lado por la necesidad de los eslabones de conocerse entre sus vecinos de cadena, y por otro lado, más relevante en cuanto al impacto sobre la aplicación, por el hecho de que el matcheo de un mensaje con una regla se hace en base del asunto y podrían existir varias reglas para un mismo patrón de asunto de mensaje. En el patrón mencionado (en su versión original), no es posible que dos eslabones puedan actuar sobre el mismo elemento.

Por lo tanto, se deja extensible la aplicación desde el punto de vista del agregado de reglas, independizándose del orden en el cual deben ser ejecutadas en una cadena de eslabones; pudiendo de esa manera agregar de manera simple Reglas para un mismo patrón.

Bugs Conocidos

Los bugs de prioridad 1 y 2 deben ser cerrados para poder considerar al código como listo para la entrega. Los bugs de prioridad 3 presentan problemas funcionales pero no impiden el uso de la aplicación. Los bugs de prioridad 4 se relacionan con la estética de la aplicación.

Bugs	Prioridad	Notas
El estado del ticket no se guarda en la base de datos.	**3**	Se necesita .NET Framework 4.5 para que Entity Framework guarde Enums en la DB. (fuente)
La aplicación Web no aplica reglas de negocio.	**3**	
La aplicación Web no es visualmente atractiva.	**4**	

Extras

Link al repositorio

http://code.google.com/p/td-20122c-tp/source/checkout